Escola de Matemática Aplicada

Aula Prática 2 - Álgebra Linear Numérica

Aluno: *Gustavo Murilo Cavalcante Carvalho*
Programa: *Bacharelado em Matemática Aplicada*
Disciplina: *Álgebra Linear Numérica*
Professor: *Antonio Carlos Saraiva Branco*
Email: *gustavomurilo012@gmail.com*

Data: *07 de abril de 2024*

Sumário

Problema 1

Implementar uma função Scilab resolvendo um sistema linear $Ax = b$ usando o algoritmo iterativo de Jacobi.

Solução:

```
function [x_k, k, norma_dif, norma_residuo] = Jacobi(A, b, x_0, E, M, tipo_norma)
% ////////////////////////////////////////////////////////////////////
% Variáveis de entrada:
% A: Matriz (n x m);
% b: vetor (n x 1);
% x_0: aproximação inicial da solução do sistema (n x 1);
% E: tolerância da aproximação;
% M: número máximo de iterações M;
% tipo_norma: indicação do tipo de norma a ser utilizada (1, 2 ou inf).
% ////////////////////////////////////////////////////////////////////
% Variáveis de saída:
% x_k: a solução obtida pelo método iterativo;
% k: número de iterações efetuadas;
% norma_dif: a norma da diferença entre as duas últimas aproximações;
% norma_residuo: a norma do resíduo (||r_k|| = ||b-Ax||).
% ////////////////////////////////////////////////////////////////////

  % Decomposição da matriz dada
  L = tril(A,-1);
  U = triu(A,1);
  D = diag(A);

  % Criação da matriz do método e do v
  inv_D = 1 ./ D;
  M_J = - inv_D .* (L+U); % Matriz de Jacobi
  v_J = inv_D .* b;

  k = 0;
  while k < M
    % Cálculo da k-ésima solução estimada pelo método iterativo
```

```
    x_k = M_J * x_0 + v_J;

    % Verifica a diferença entre a solução atual e a anterior
    norma_dif = norm(x_k-x_0, tipo_norma);

    % Atualização da condição de parada
    if norma_dif < E
      break; % Termina a execução da estrutura de repetição
    end

    % Atualiza a solução anterior, preparando para a próxima iteração
    x_0 = x_k;
    % Atualiza o contador de iterações
    k = k + 1;
  end

  norma_residuo = norm(b-A*x_k, tipo_norma);
end
```

Problema 2

Implementar uma função Scilab resolvendo um sistema linear Ax = b usando o algoritmo iterativo de Gauss-Seidel.

a) Usando a função inv() do MATLAB.

Solução:

```
function [x_k, k, norma_dif, norma_residuo] = Gauss_Seidel_a(A, b, x_0, E, M, tipo_norma)
% ////////////////////////////////////////////////////////////////////
% Variáveis de entrada:
% A: Matriz (n x m);
% b: vetor (n x 1);
% x_0: aproximação inicial da solução do sistema (n x 1);
% E: tolerância da aproximação;
% M: número máximo de iterações M;
% tipo_norma: indicação do tipo de norma a ser utilizada (1, 2 ou inf).
% ////////////////////////////////////////////////////////////////////
% Variáveis de saída:
% x_k: a solução obtida pelo método iterativo;
% k: número de iterações efetuadas;
% norma_dif: a norma da diferença entre as duas últimas aproximações;
% norma_residuo: a norma do resíduo (||r_k|| = ||b-Ax||).
% ////////////////////////////////////////////////////////////////////

  % Decomposição da matriz dada
  LD = tril(A);  % L+D
  U = triu(A,1);

  % Criação da matriz do método
  inv_LD = inv(LD);
  M_GS = - inv_LD * U; % Matriz de Gauss-Seidel
  v_GS = inv_LD * b;

  k = 0;
  while k < M
    % Cálculo da k-ésima solução estimada pelo método iterativo
```

```
    x_k = M_GS * x_0 + v_GS;

    % Verifica a diferença entre a solução atual e a anterior
    norma_dif = norm(x_k-x_0, tipo_norma);

    % Atualização da condição de parada
    if norma_dif < E
      break; % Termina a execução da estrutura de repetição
    end

    % Atualiza a solução anterior, preparando para a próxima iteração
    x_0 = x_k;
    % Atualiza o contador de iterações
    k = k + 1;
  end

  norma_residuo = norm(b-A*x_k, tipo_norma);
end
```

b) Resolvendo o sistema $(L + D)\, x_{k+1} = U\, x_k + b$ em cada iteração.

Solução:

```
function [x_k, k, norma_dif, norma_residuo] = Gauss_Seidel_b(A, b, x_0, E, M, tipo_norma)
% ////////////////////////////////////////////////////////////////////
% Variáveis de entrada:
% A: Matriz (n x m);
% b: vetor (n x 1);
% x_0: aproximação inicial da solução do sistema (n x 1);
% E: tolerância da aproximação;
% M: número máximo de iterações M;
% tipo_norma: indicação do tipo de norma a ser utilizada (1, 2 ou inf).
% ////////////////////////////////////////////////////////////////////
% Variáveis de saída:
% x_k: a solução obtida pelo método iterativo;
% k: número de iterações efetuadas;
% norma_dif: a norma da diferença entre as duas últimas aproximações;
% norma_residuo: a norma do resíduo (||r_k|| = ||b-Ax||).
% ////////////////////////////////////////////////////////////////////

  % Decomposição da matriz dada
  LD = tril(A);  % L+D
  U = triu(A,1);

  k = 0;
  while k < M
    % Cálculo da k-ésima solução estimada pelo método iterativo
    % Resolvendo (L+D)*x_k = -U*x_(k-1) + b
    v_GS = -U * x_0 + b;
    x_k = Resolve_L(LD, v_GS);

    norma_dif = norm(x_k - x_0, tipo_norma);

    % Atualização da condição de parada
```

```
    if norma_dif < E
      break; % Termina a execução da estrutura de repetição
    end

    % Atualiza a solução anterior, preparando para a próxima iteração
    x_0 = x_k;
    % Atualiza o contador de iterações
    k = k + 1;
  end

  norma_residuo = norm(b-A*x_k, tipo_norma);
end


function [x] = Resolve_L(L,b)
  n = size(L,1);
  x = zeros(n,1);

  x(1) = b(1) / L(1,1);
  for i = 2 : n
    x(i) = (b(i) - L(i, 1 : i-1) * x(1 : i-1)) / L(i,i);
  end
end
```

Aviso:

As duas funções do "Método de Gauss_Seidel" implementadas com as instruções do "Problema 2" têm os mesmo resultados em todos os testes feitos nos demais problemas. Por isso, quando solicitado o uso de uma função que resolva por esse método sem especificar qual, evito colocar as duas funções, pois verificamos que ao nível de resultado elas são idênticas. A diferença entre elas se dá no tempo de execução, o que é conferido no "Problema 6".

Problema 3

Teste as funções implementadas para resolver o sistema

$$\begin{cases} x - 4y + 2z = 2 \\ \quad 2y + 4z = 1 \\ 6x - \ y - 2z = 1 \end{cases}$$

Use o vetor $x_0 = (0, 0, 0)$ como aproximação inicial.

Agora reordene as equações do sistema dado, de modo que a matriz dos coeficientes seja estritamente diagonal dominante e teste novamente as funções implementadas. Comente os resultados.

Solução:

Para podermos testar as funções cridas para os itens anteriores, temos que escrever o sistema em forma matricial.

```
>> disp(A_1)              >> disp(b_1)      >> disp(x_0)
     1    -4     2             2                 0
     0     2     4             1                 0
     6    -1    -2             1                 0
```

Testes iniciais

Seguem os testes abaixo feitos sob as mesmas condições para cada função.

Teste 1 - Jacobi:

```
>> [x_k, k, norma_dif, norma_residuo] = Jacobi(A_1, b_1, x_0, 1e-6, 100, 2)

x_k =                     k =               norma_dif =
   2.1035e+50                100               3.3588e+50
   1.8518e+50
  -1.8183e+50                              norma_residuo =
                                               1.7326e+51
```

Teste 2 - Gauss_Seidel_a:

```
>> [x_k, k, norma_dif, norma_residuo] = Gauss_Seidel_a(A_1, b_1, x_0,
 ↪  1e-6, 100, 2)

x_k =                     k =               norma_dif =
   2.936e+67                 100               3.9695e+68
  -3.0252e+68
   2.3934e+68                              norma_residuo =
                                               1.7538e+69
```

Teste 3 - Gauss_Seidel_b:

```
>> [x_k, k, norma_dif, norma_residuo] = Gauss_Seidel_b(A_1, b_1, x_0,
↪  1e-6, 100, 2)
```

```
x_k =                   k =            norma_dif =
   2.936e+67               100            3.9695e+68
  -3.0252e+68
   2.3934e+68                          norma_residuo =
                                          1.7538e+69
```

Os métodos obtiveram resultados diferentes, mas as funções do método de Gauss-Seidel foram coerentes entre si. Em todos os teste, foi atingido o número máximo de iterações (100 no caso) e a norma do resíduo é muito grande (escala de 10^{70}) o que levanta a suspeita de que a matriz em questão não seja convergente.

O "Critério das Linhas" é válido para verificar validade de ambos os métodos, ele estabelece a seguinte relação: se a diagonal de uma matriz A é estritamente dominante, então os métodos de Jacobi e Gauss-Seidel convergem para a solução. Para que uma matriz seja estritamente dominante, é preciso que a seguinte desigualdade seja verdade para todas as linhas da matriz: $\sum_{j=1,j\neq i}^{n} |a_{ij}| < |a_{ii}|$.

Vemos que a matriz A_1 não satisfaz esse critério:

$$A_1 = \begin{bmatrix} 1 & -4 & 2 \\ 0 & 2 & 4 \\ 6 & -1 & -2 \end{bmatrix} \implies \begin{matrix} |4| + |2| = 6 \not< |1| \\ |0| + |4| = 4 \not< |2| \\ |6| + |-1| = 7 \not< |-2| \end{matrix} \tag{1}$$

O "Critério de Sassenfeld" é condição suficiente para a validade do método iterativo de Guass-Seidel para uma matriz A. Para atender esse critério, é preciso que $\forall i \in 1, 2, ..., n$, $\beta_i = \dfrac{1}{|a_{ii}|}(\sum_{j=1}^{i-1} |a_{ij}|\beta_j + \sum_{j=i+1}^{n} |a_{ij}|) < 1$.

Ao verificar, vemos que a matriz A_1 também não satisfaz esse critério, pois $\exists \beta_i \not< 1$.

$$\begin{aligned} \beta_1 &= \frac{1}{|1|}(|-4| + |2|) = 6 \not< 1 \\ \beta_2 &= \frac{1}{|2|}(|0|\,\frac{1}{6} + |2|) = 1 \not< 1 \\ \beta_3 &= \frac{1}{|2|}(|6|\,\frac{1}{6} + |-1|\,\frac{1}{2}) = \frac{3}{4} < 1 \end{aligned} \tag{2}$$

Como a matriz A_1 não satisfaz nenhum dos critérios acima, os métodos iterativos podem ou não convergir para a solução. Pelos fatores listados, ainda acho que esses métodos iterativos não são eficazes para essa matriz.

Testes com a matriz com diagonal estritamente dominante

Alteramos a ordem das linhas A_1 para que ela tenha diagonal estritamente dominante, chamamos o resultado de A_2, uma matriz que satisfaz o critério das linhas, e, portanto, certamente os métodos iterativos com os quais estamos trabalhando certamente convergem para uma solução. Fazemos com b_1 a mesma permutação feita com A_1, obtendo o vetor b_2.

$$b_2 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \qquad \Bigg| \qquad A_2 = \begin{bmatrix} 6 & -1 & -2 \\ 1 & -4 & 2 \\ 0 & 2 & 4 \end{bmatrix} \implies \begin{matrix} |-1| + |-2| = 3 < |6| \\ |1| + |2| = 3 < |-4| \\ |0| + |2| = 2 < |4| \end{matrix} \tag{3}$$

Teste 1 - Jacobi:

```
>> [x_k, k, norma_dif, norma_residuo] = Jacobi(A_2, b_2, x_0, 1e-6, 100, 2)

x_k =                    k =              norma_dif =
    0.25                     19               6.8689e-07
  -0.25
    0.375                                 norma_residuo =
                                              2.0961e-06
```

Teste 2 - Gauss_Seidel_a:

```
>> [x_k, k, norma_dif, norma_residuo] = Gauss_Seidel_a(A_2, b_2, x_0,
↪  1e-6, 100, 2)

x_k =                    k =              norma_dif =
   0.25                      11               2.7767e-07
  -0.25
   0.375                                  norma_residuo =
                                              2.4835e-07
```

Teste 3 - Gauss_Seidel_b:

```
>> [x_k, k, norma_dif, norma_residuo] = Gauss_Seidel_b(A_2, b_2, x_0,
↪  1e-6, 100, 2)

x_k =                    k =              norma_dif =
   0.25                      11               2.7767e-07
  -0.25
   0.375                                  norma_residuo =
                                              2.4835e-07
```

Para todas as funções, a solução é basicamente a mesma, a norma do resíduo é ínfima, na escala de 10^{-6} ou 10^{-7}, então obtemos uma solução muito próxima da real. O método de Gauss_Seidel é vantajoso nesse caso, pois obtém mais precisão em menos iterações.

Problema 4

a) Para o sistema do exercício 3 da Lista de Exercícios 2, mostre que o método de Jacobi com $x^{(0)} = 0$ falha em dar uma boa aproximação após 25 iterações.

Solução:

O sistema em questão ($A_3\, x_3 = b_3$) é dado por:

$$A_3 = \begin{bmatrix} 2 & -1 & 1 \\ 2 & 2 & 2 \\ -1 & -1 & 2 \end{bmatrix} \qquad x_3 = \begin{bmatrix} 1 \\ 2 \\ -1 \end{bmatrix} \qquad b_3 = \begin{bmatrix} -1 \\ 4 \\ -5 \end{bmatrix}$$

Fazendo um teste do método de Jacobi com A_3, b_3 e $x_0 = (0, 0, 0)$ limitado a 25 25 iterações obtemos o resultado abaixo. Dele vemos que o máximo de iterações foi alcançado, ainda assim a norma do resíduo é bem grande, o que significa que a solução obtida não é próxima do que deveria ser, o resultado é ruim.

```
>> [x_k, k, norma_dif, norma_residuo] = Jacobi(A_3, b_3, x_0, 1e-6,
↪  25, 2)
```

```
x_k =                    k =            norma_dif =
  -20.828                  25             48.219
    2
  -22.828                               norma_residuo =
                                          111.3
```

É fácil ver que o "Critério das Linhas", não é satisfeito por A_3, na segunda linha o módulo da diagonal é menor que a soma do módulo dos demais elementos, $|2| < |2| + |2|$.

Outro indício para a não convergência do método é o raio espectral da matriz do método de Jacobi para A_3, $\rho(M_{J3}) = 1.118 > 1$.

```
>> M_J_3 = - inv(triu(tril(A_3))) * (triu(A_3,1) + tril(A_3,-1))
M_J_3 =
     0      0.5   -0.5
    -1      0     -1
     0.5    0.5    0


>> rho_3 = max(abs(eig(M_J_3)))
rho_3 =
     1.118
```

b) Use o método de Gauss-Seidel com $x^{(0)} = 0$ para obter uma aproximação da solução do sistema linear com precisão de 10^{-5} na norma-infinito.

Solução:

Como podemos ver abaixo, matriz A_3 satisfaz o "Critério de Sassenfeld", não é satisfeito por A_3. Verificamos então a convergência da matriz do método de Gauss_Seild para A_3, que decerto ocorre se $\rho(M_G S_3) < 1$. Vemos no teste abaixo que ela é convergente.

```
>> M_GS_3 = -inv(tril(A_3)) * triu(A_3, 1)
M_GS_3 =
     0     0.5   -0.5
     0    -0.5   -0.5
     0     0     -0.5

>> rho_GS_3 = max(abs(eig(M_GS_3)))
rho_GS_3 =
     0.5
```

Fazendo o teste com as condições indicadas, obtemos a solução esperada $x_k = (1, 2, -1)$ com erro muito pequeno e em poucas iterações (22).

```
>> [x_k, k, norma_dif, norma_residuo] = Gauss_Seidel_b(A_3, b_3, x_0,
↪ 1e-5, 25, inf)

x_k =                       k =                norma_dif =
   1                           22                 7.2718e-06
   2
  -1                                           norma_residuo =
                                                  6.9141e-06
```

Problema 5

a) Utilize o método iterativo de Gauss-Seidel para obter uma aproximação da solução do sistema linear do exercício 5 da Lista de Exercícios 2, com tolerância de 10^{-2} e o máximo de 300 iterações.

Solução:

Seja ($A_4\, x_4 = b_4$) o sistema do exercício 5, as matrizes são:

$$A_4 = \begin{bmatrix} 2 & 0 & -1 \\ -\frac{1}{2} & 1 & -\frac{1}{4} \\ 1 & -\frac{1}{2} & 1 \end{bmatrix} \qquad x_4 = \begin{bmatrix} 0,9 \\ -0,8 \\ 0,7 \end{bmatrix} \qquad b_4 = \begin{bmatrix} 0,2 \\ -1,45 \\ 2 \end{bmatrix}$$

Uma solução bem próxima é obtida em 12 iterações. Embora não satisfaça o "Critério das Linhas", essa matriz satisfaz o "Critério de Sassenfeld", então é esperado que as funções que usam o método de Gauss_Seidel sejam capazes de achar uma solução rapidamente.

```
>> [x_k, k, norma_dif, norma_residuo] = Gauss_Seidel_b(A_4, b_4, x_0,
↪  1e-2, 300, inf)
```

```
x_k =                   k =           norma_dif =

   0.89751                12             0.0064659
  -0.80187
   0.701551                           norma_residuo =

                                         0.0040412
```

Critério de Sassenfeld para A_4:

$$\begin{gathered} \frac{1}{|2|}\Big(|0| + |-1|\Big) = \frac{1}{2} < 1 \\ \frac{1}{|1|}\left(\left|\frac{-1}{2}\right|\frac{1}{2} + \left|\frac{-1}{4}\right|\right) = \frac{1}{2} < 1 \\ \frac{1}{|1|}\left(|1|\,\frac{1}{2} + \left|\frac{-1}{2}\right|\frac{1}{2}\right) = \frac{3}{4} < 1 \end{gathered} \tag{4}$$

b) O que acontece ao repetir o item a) quando o sistema é alterado para:

$$\begin{cases} x_1 \qquad\quad - 2\,x_3 = 0,2 \\ -\dfrac{1}{2}x_1 + \quad x_2 - \dfrac{1}{4}x_3 = -1,425 \\ x_1 - \dfrac{1}{2}x_2 + \quad x_3 = 2 \end{cases}$$

Solução:

O novo sistema é:

$$A_5 = \begin{bmatrix} 1 & 0 & -2 \\ -\dfrac{1}{2} & 1 & -\dfrac{1}{4} \\ 1 & -\dfrac{1}{2} & 1 \end{bmatrix} \qquad x_5 = \begin{bmatrix} 0,9 \\ -0,8 \\ 0,7 \end{bmatrix} \qquad b_5 = \begin{bmatrix} 0,2 \\ -1,45 \\ 2 \end{bmatrix}$$

Ao tentar resolver esse novo sistema com ambos os métodos obtemos péssimos resultado com erros enormes, as 300 iterações são atingidas e só servem para distanciar x_k da solução, as matrizes dos dois métodos divergem da solução.

Teste 1 - Gauss_Seidel_a:

```
>> [x_k, k, norma_dif, norma_residuo] = Gauss_Seidel_a(A_5, b_5, x_0,
↪  1e-2, 300, inf)
```

```
x_k =                     k =              norma_dif =

   2.1569e+41                300              3.7255e+41
   1.348e+41
  -1.4829e+41                              norma_residuo =

                                              5.1226e+41
```

Teste 2 - Jacobi:

```
>> [x_k, k, norma_dif, norma_residuo] = Jacobi(A_5, b_5, x_0, 1e-2,
↪  300, inf)
```

```
x_k =                     k =              norma_dif =

   1.3448e+43                300              3.137e+43
  -7.2803e+42
   1.5665e+43                              norma_residuo =

                                              3.2753e+43
```

Podemos dizer que essa matriz é mal condicionada por se tornar instável com a troca de 2 elementos, pois essa pequena mudança é suficiente para interromper a convergência dos métodos iterativos.

Problema 6

Agora gere matrizes $A_{n\,x\,n}$ com diagonal estritamente dominante para $n = 10$, $n = 100$, $n = 1000$, $n = 2000$, ... bem como vetores b com dimensões compatíveis e resolva esses sistemas Ax=b pelo Método de Gauss-Seidel, usando as duas versões implementadas no item 2. Use as funções tic() e toc() do Scilab para medir os tempos de execução e compará-los.

Solução:

Perante a necessidade de testar o tempo de execução das funções "Gauss_Seidel_a" e "Gauss_Seidel_b", criei uma função para calcular esse tempo usando as funções tic e toc, e um script com um teste automatizado para matrizes de diferentes dimensões.

Para criar uma matriz $A_{n\,x\,n}$ com diagonal estritamente dominante, começo criando um matriz com números inteiros de 1 a 5. A soma de cada linha é menor ou igual a $5n$, bem como cada elemento da diagonal é maior ou igual a 1, portanto basta somar $5n$ a cada elemento da diagonal para garantir que a matriz seja estritamente dominante.

Criada a matriz, a multiplicamos por um vetor x formado por inteiros aleatórios de 1 a 9, resultando em um vetor b. Assim, temos um sistema $A\,x = b$.

Consideramos o vetor inicial $x_0 = \vec{0}$.

Dispondo disso, podemos testar o tempo de execução de cada função. Para automatizar esses teste, criei um script que armazena o tempo de execução de cada uma das funções para matrizes de dimensões {10, 100, 1.000, 2.000, 5.000, 10.000, 15.000, 20.000}.

Sobre os resultados, vi que para matrizes de dimensões pequenas, não há tanta diferença, mas no geral a função Gauss_Seidel_b leva menos tempo. Já para dimensões maiores a diferença é significativa, resolver o sistema é muito mais rápido para matrizes de dimensão na escala de 10^4 ou maior, tendo mais que o dobro da velocidade da função que inverte uma matriz.

```
function [tempo] = Testar_Velocidade(n, func)
  tic; % Inicia o contador de tempo

  [A, b] = Criar_Sistema(n);
  x_0 = zeros(n,1);

  if func == 'a'
    Gauss_Seidel_a(A, b, x_0, 1e-6, 500, 2);
  elseif func == 'b'
```

```
    Gauss_Seidel_b(A, b, x_0, 1e-6, 500, 2);
  end

  tempo = toc; % Finaliza a contagem do tempo
end


function [A, b] = Criar_Sistema(n)
  % Cria uma matriz com diagonal estritamente dominante
  A = Criar_Dominante(n);
  x = randi(9, n, 1);
  b = A*x;
end


function [A] = Criar_Dominante(n)
  % Cria uma matriz nxn com inteiros aleatórios de 1 a 5
  A = randi(5, n, n);

  % A soma máxima de uma linha é 5*(n-1). Para tornar a matriz
  ↪ estritamente dominante, basta somar 5*n aos elementos da diagonal
  for i = 1 : n
    A(i,i) = A(i,i) + 5*n;
  end
end
```

Script criado para automatizar os testes:

```
n = [10, 100, 1000, 2000, 5000, 10000, 15000, 20000];
velocidade_a = zeros(1, 8);
velocidade_b = zeros(1, 8);

for i = 1 : 8
  velocidade_a(i) = Testar_Velocidade(n(i), 'a');
  velocidade_b(i) = Testar_Velocidade(n(i), 'b');
end

diferenca = velocidade_a - velocidade_b;
```

Resultados dos testes:

Variables - velocidade_b

velocidade_b

1x8 double

	1	2	3	4	5	6	7	8
1	7.3400e-04	0.0027	0.0763	0.2285	1.8970	9.5878	22.3331	49.7909
2								

velocidade_a

1x8 double

	1	2	3	4	5	6	7	8
1	0.0132	0.0034	0.1086	0.6604	2.5008	17.3375	47.1714	118.7122
2								

diff

1x8 double

	1	2	3	4	5	6	7	8
1	0.0125	6.9300e-...	0.0323	0.4319	0.6038	7.7497	24.8383	68.9213
2								

Variables - velocidade_b

velocidade_b

1x8 double

	1	2	3	4	5	6	7	8
1	0.0011	0.0021	0.0678	0.1969	1.5470	8.1823	19.7264	45.3368
2								

velocidade_a

1x8 double

	1	2	3	4	5	6	7	8
1	0.0028	0.0024	0.0673	0.3207	2.0742	13.8262	45.3271	116.1901
2								

diff

1x8 double

	1	2	3	4	5	6	7	8
1	0.0017	3.0600e-...	-5.1000e...	0.1238	0.5272	5.6438	25.6007	70.8533
2								